UNIVERSITY of York

This is a repository copy of *Calculational Verification of Reactive Programs with Reactive Relations and Kleene Algebra*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/134042/

Version: Accepted Version

Proceedings Paper:
Foster, Simon David orcid.org/0000-0002-9889-9514, Ye, Kangfeng, Cavalcanti, Ana Lucia Caneca orcid.org/0000-0002-0831-1976 et al. (1 more author) (2018) Calculational Verification of Reactive Programs with Reactive Relations and Kleene Algebra. In: Guttmann, Walter, Desharnais, Jules and Joosten, Stef, (eds.) Relational and Algebraic Methods in Computer Science - 17th International Conference, RAMiCS 2018, Proceedings. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) . Springer , pp. 205-224.

https://doi.org/10.1007/978-3-030-02149-8_13

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Calculational Verification of Reactive Programs with Reactive Relations and Kleene Algebra

Simon Foster[ORCiD], Kangfeng Ye, Ana Cavalcanti, and Jim Woodcock

University of York
simon.foster@york.ac.uk

Abstract. Reactive programs are ubiquitous in modern applications, and so verification is highly desirable. We present a verification strategy for reactive programs with a large or infinite state space utilising algebraic laws for reactive relations. We define novel operators to characterise interactions and state updates, and an associated equational theory. With this we can calculate a reactive program's denotational semantics, and thereby facilitate automated proof. Of note is our reasoning support for iterative programs with reactive invariants, which is supported by Kleene algebra. We illustrate our strategy by verifying a reactive buffer. Our laws and strategy are mechanised in Isabelle/UTP, which provides soundness guarantees, and practical verification support.

1 Introduction

Reactive programming [1, 2] is a paradigm that enables effective description of software systems that exhibit both internal sequential behaviour and event-driven interaction with a concurrent party. Reactive programs are ubiquitous in safety-critical systems, and typically have a very large or infinite state space. Though model checking is an invaluable verification technique, it exhibits inherent limitations with state explosion and infinite-state systems that can be overcome by supplementing it with theorem proving.

Previously [3], we have shown how *reactive contracts* support automated proof. They follow the design-by-contract paradigm [4], where programs are accompanied by pre- and postconditions. Reactive programs are often non-terminating and so we also capture intermediate behaviours, where the program has not terminated, but is quiescent and offers opportunities to interact. Our contracts are triples, $[P_1 \vdash P_2 \mid P_3]$, where P_1 is the precondition, P_3 the postcondition, and P_2 the "pericondition". P_2 characterises the quiescent observations in terms of the interaction history, and the events enabled at that point.

Reactive contracts describe communication and state updates, so P_1, P_2, and P_3 can refer to both a trace history of events and internal program variables. They are, therefore, called "reactive relations": like relations that model sequential programs, they can refer to variables before (x) and later (x') in execution, but also the interaction trace (tt), in both intermediate and final observations.

Verification using contracts employs refinement ($\sqsubseteq$), which requires that an implementation weakens the precondition, and strengthens both the peri-

and postcondition when the precondition holds. We employ the "programs-as-predicates" approach [5], where the implementation (Q) is itself denoted as a composition of contracts. Thus, a verification problem, $[P_1 \vdash P_2 \mid P_3] \sqsubseteq Q$, can be solved by calculating a program $[Q_1 \vdash Q_2 \mid Q_3] = Q$, and then discharging three proof obligations: (1) $Q_1 \sqsubseteq P_1$; (2) $P_2 \sqsubseteq (Q_2 \wedge P_1)$; and (3) $P_3 \sqsubseteq (Q_3 \wedge P_1)$. These can be further decomposed, using relational calculus, to produce verification conditions. In [3] we employ this strategy in an Isabelle/HOL tactic.

For reactive programs of a significant size, these relations are complex, and so the resulting proof obligations are difficult to discharge using relational calculus. We need, first, abstract patterns so that the relations can be simplified. This necessitates bespoke operators that allow us to concisely formulate the different kinds of observation. Second, we need calculational laws to handle iterative programs, which are only partly handled in our previous work [3].

In this paper we present a novel calculus for description, composition, and simplification of reactive relations in the stateful failures-divergences model [6,7]. We characterise conditions, external interactions, and state updates. An equational theory allows us to reduce pre-, peri-, and postconditions to compositions of these atoms using operators of Kleene algebra [8] (KA) and utilise KA proof techniques. Our theory is characterised in the Unifying Theories of Programming [6,9] (UTP) framework. For that, we identify a class of UTP theories that induce KAs, and utilise it in derivation of calculational laws for iteration. We use our UTP mechanisation, called Isabelle/UTP [10], to implement an automated verification approach for infinite-state systems with rich data structures.

The paper is structured as follows. §2 outlines preliminary material. §3 identifies a class of UTP theories that induce KAs, and applies this for calculation of iterative contracts. §4 specialises reactive relations with new atomic operators to capture stateful failures-divergences, and derives their equational theory. §5 extends this with support for calculating external choices. §6 completes the theoretical picture with while loops and reactive invariants. §7 demonstrates the resulting proof strategy in a small verification. §8 outlines related work and concludes. All our theorems have been mechanically verified in Isabelle/UTP[1] [10–13].

2 Preliminaries

Kleene Algebras [8] (KA) characterise sequential and iterative behaviour in nondeterministic programs using a signature $(K, +, 0, \cdot, 1, ^*)$, where $+$ is a choice operator with unit 0, and $\cdot$ a composition operator, with unit 1. Kleene closure P^* denotes iteration of P using $\cdot$ zero or more times. We consider the class of weak Kleene algebras [14], which build on weak dioids.

Definition 2.1. *A weak dioid is a structure* $(K, +, 0, \cdot, 1)$ *such that* $(S, +, 0)$ *is an idempotent and commutative monoid;* $(S, \cdot, 1)$ *is a monoid;* $\cdot$ *left- and right-distributes over* $+$*; and* 0 *is a left annihilator for* $\cdot$*.*

[1] All proofs can be found in the cited series of Isabelle/HOL reports. For historical reasons, we use the syntax $\boldsymbol{R}_s(P \vdash Q \diamond R)$ in our mechanisation for a contract $[P \vdash Q \mid R]$, which builds on Hoare and He's original syntax for the theory of designs [6].

The 0 operator represents miraculous behaviour. It is a left annihilator of composition, but not a right annihilator as this often does not hold for programs. K is partially ordered by $x \leq y \triangleq (x + y = y)$, which is defined in terms of $+$, and has least element 0. A weak KA extends this with the behaviour of the star.

Definition 2.2. *A weak Kleene algebra is a structure* $(K, +, 0, \cdot, 1, ^*)$ *such that*

1. $(K, +, 0, \cdot, 1)$ *is a weak dioid*
2. $1 + x \cdot x^* \leq x^*$
3. $z + x \cdot y \leq y \Rightarrow x^* \cdot z \leq y$
4. $z + y \cdot x \leq y \Rightarrow z \cdot x^* \leq y$

Various enrichments and specialisations of these axioms exist; for a complete survey see [8]. For our purposes, these axioms alone suffice. From this base, a number of useful identities can be derived, some of which are listed below.

Theorem 2.3. $x^{**} = x^* \quad x^* = 1 + x \cdot x^* \quad (x + y)^* = (x^* \cdot y^*)^* \quad x \cdot x^* = x^* \cdot x$

UTP [6, 9] uses the "programs-as-predicate" approach to encode denotational semantics and facilitate reasoning about programs. It uses the alphabetised relational calculus, which combines predicate calculus operators like disjunction ($\vee$), complement ($\neg$), and quantification ($\exists\, x \bullet P(x)$), with relation algebra, to denote programs as binary relations between initial variables (x) and their subsequent values (x'). The set of relations *Rel* is partially ordered by refinement $\sqsubseteq$ (refined-by), denoting universally closed reverse implication, where $\mathbf{false}$ refines every relation. Relational composition ($\fatsemi$) denotes sequential composition with identity $\mathbb{I}$. We summarise the algebraic properties of relations below.

Theorem 2.4. $(Rel, \sqsupseteq, \mathbf{false}, \fatsemi, \mathbb{I})$ *is a Boolean quantale [15], so that:*

1. $(Rel, \sqsubseteq)$ *is a complete lattice, with infimum* $\bigvee$*, supremum* $\bigwedge$*, greatest element* $\mathbf{false}$*, least element* $\mathbf{true}$*, and weakest (least) fixed-point operator* $\mu\, F$*;*
2. $(Rel, \vee, \mathbf{false}, \wedge, \mathbf{true}, \neg)$ *is a Boolean algebra;*
3. $(Rel, \fatsemi, \mathbb{I})$ *is a monoid with* $\mathbf{false}$ *as left and right annihilator;*
4. $\fatsemi$ *distributes over* $\bigvee$ *from the left and right.*

We often use $\bigsqcap_{i \in I} P(i)$ to denote an indexed disjunction over I, which intuitively refers to a nondeterministic choice. Note that the partial order $\leq$ of the Boolean quantale is $\sqsupseteq$, and so our lattice operators are inverted: for example, $\bigvee$ is the infimum with respect to $\sqsubseteq$, and $\mu\, F$ is the least fixed-point.

Relations can directly express sequential programs, whilst enrichment to characterise more advanced paradigms — such as object orientation [16], real-time [17], and concurrency [6] — can be achieved using UTP theories. A UTP theory is characterised as the set of fixed-points of a function $\mathbf{H} : Rel \to Rel$, called a healthiness condition. If P is a fixed-point of $\mathbf{H}$ it is said to be $\mathbf{H}$-healthy, and the set of healthy relations is $[\![\mathbf{H}]\!] \triangleq \{P \mid \mathbf{H}(P) = P\}$. In UTP, it is desirable that $\mathbf{H}$ is idempotent and monotonic so that $[\![\mathbf{H}]\!]$ forms a complete lattice under $\sqsubseteq$, and thus reasoning about both nondeterminism and recursion is possible.

Theory engineering and verification of programs using UTP is supported by Isabelle/UTP [10], which provides a shallow embedding of the relational calculus on top of Isabelle/HOL, and various approaches to automated proof. In this paper, we use a UTP theory to characterise reactive programs.

Reactive Programs. Whilst sequential programs determine the relationship between an initial and final state, reactive programs also pause during execution to interact with the environment. For example, the CSP [9, 18] and *Circus* [7] languages can model networks of concurrent processes that communicate using shared channels. Reactive behaviour is described using primitives such as event prefix $a \rightarrow P$, which awaits event a and then enables P; conditional guard, $b \mathbin{\&} P$, which enables P when b is true; external choice $P \Box Q$, where the environment resolves the choice by communicating an initial event of P or Q; and iteration $\textbf{\textit{while}}\ b\ \textbf{\textit{do}}\ P$. Channels can carry data, and so events can take the form of an input $(c?x)$ or output $(c!v)$. *Circus* processes also have local state variables that can be assigned $(x := v)$. We exemplify *Circus* with an unbounded buffer.

Example 2.5. In the *Buffer* process below, variable $bf : \mathrm{seq}\,\mathbb{N}$ records the elements, and channels $inp(n : \mathbb{N})$ and $outp(n : \mathbb{N})$ represent inputs and outputs.

$$Buffer \triangleq bf := \langle\rangle \fatsemi \left(\begin{array}{l} \textbf{\textit{while}}\ true\ \textbf{\textit{do}} \\ \quad \left(\begin{array}{l} inp?v \rightarrow bf := bf \frown \langle v \rangle \\ \Box\ (\#bf > 0)\ \&\ out!(head(bf)) \rightarrow bf := tail(bf) \end{array} \right) \end{array} \right)$$

Variable bf is set to the empty sequence $\langle\rangle$, and then a non-terminating loop describes the main behaviour. Its body repeatedly allows the environment to either provide a value v over inp, followed by which bf is extended, or else, if the buffer is non-empty, receive the value at the head, and then bf is contracted. □

The semantics of such programs can be captured using reactive contracts [3]:

$$[\, P_1(tt, st, r) \vdash P_2(tt, st, r, r') \mid P_3(tt, st, st', r, r') \,]$$

Here, $P_{1\ldots3}$ are reactive relations that respectively encode, (1) the precondition in terms of the initial state and permissible traces; (2) permissible intermediate interactions with respect to an initial state; and (3) final states following execution. Pericondition P_2 and postcondition P_3 are both within the "guarantee" part of the underlying design contract, and so must be strengthened by refinment; see Appendix A and [3] for details. P_2 does not refer to intermediate state variables since they are concealed when a program is quiescent.

Variable tt refers to the trace, and $st, st' : \Sigma$ to the state, for state space Σ. Traces are equipped with operators for the empty trace $\langle\rangle$, concatenation $tt_1 \frown tt_2$, prefix $tt_1 \leq tt_2$, and difference $tt_1 - tt_2$, which removes a prefix tt_2 from tt_1. Technically, tt is not a relational variable, but an expression $tt \triangleq tr' - tr$ where tr, tr', as usual in UTP, encode the trace relationally [6]. Nevertheless, due to our previous results [10, 19], tt can be treated as a variable. Here, traces are modelled as finite sequences, $tt : \mathrm{seq}\, Event$, for some event set, though other models are also admitted [19]. Events can be parametric, written $a.x$, where a is a channel and x is the data. Moreover, the relations can encode additional semantic data, such as refusals, using variables r, r'. Our theory, therefore, provides an extensible denotational semantic model for reactive and concurrent languages.

To exemplify, we consider the event prefix and assignment operators from *Circus*, which require that we add variable $ref' : \mathbb{P}(Event)$ to record refusals.

$$a \rightarrow \textbf{\textit{Skip}} \triangleq [\, \textbf{\textit{true}}_r \vdash tt = \langle\rangle \wedge a \notin ref' \mid tt = \langle a \rangle \wedge st' = st \,]$$

$$x := v \;\triangleq\; [\, \mathit{true}_r \vdash \mathit{false} \mid st' = st \oplus \{x \mapsto v\} \wedge tt = \langle\rangle \,]$$

Prefix has a true precondition, indicated using the reactive relation true_r, since the environment cannot cause errors. In the pericondition, no events have occurred ($tt = \langle\rangle$), but a is not being refused. In the postcondition, the trace is extended by a, and the state is unchanged. Assignment also has a true precondition, but a false pericondition since it terminates without interaction. The postcondition updates the state, and leaves the trace unchanged.

Reactive relations and contracts are characterised by healthiness conditions ***RR*** and ***NSRD***, respectively, which we have previously described [3], and reproduce in Appendix A. ***NSRD*** specialises the theory of reactive designs [7, 9] to *normal stateful reactive designs* [3]. Both $[\![\mathit{RR}]\!]$ and $[\![\mathit{NSRD}]\!]$ are closed under sequential composition, and have units II_r and II_R, respectively. Both also form complete lattices under $\sqsubseteq$, with top elements ***false*** and ***Miracle*** $= [\mathit{true}_r \vdash \mathit{false} \mid \mathit{false}]$, respectively. ***Chaos*** $= [\mathit{false} \vdash \mathit{false} \mid \mathit{false}]$, the least determinisitic contract, is the bottom of the reactive contract lattice. We define the conditional operator $P \lhd b \rhd Q \triangleq ((b \wedge P) \vee (\neg b \wedge Q))$, where b is a condition on unprimed state variables, which can be used for both reactive relations and contracts. We then define the state test operator $[b]^{\top}_r \triangleq \mathit{II}_r \lhd b \rhd \mathit{false}$.

Contracts can be composed using relational calculus. The following identities [3, 12] show how this entails composition of the underlying pre-, peri-, and postconditions for $\bigsqcap$ and $;$, and also demonstrates closure under these operators.

Theorem 2.6 (Reactive Contract Composition).

$$\textstyle\bigsqcap_{i \in I} [P(i) \vdash Q(i) \mid R(i)] = \big[\, \bigwedge_{i \in I} P(i) \vdash \bigvee_{i \in I} Q(i) \;\big|\; \bigvee_{i \in I} R(i) \,\big] \tag{1}$$

$$[P_1 \vdash P_2 \mid P_3] \fatsemi [Q_1 \vdash Q_2 \mid Q_3] = [P_1 \wedge (P_3 \;\mathit{wp}_r\; Q_1) \vdash P_2 \vee (P_3 \fatsemi Q_2) \mid P_3 \fatsemi Q_3] \tag{2}$$

Nondeterministic choice requires all preconditions, and asserts that one of the peri- and postcondition pairs hold. For sequential composition, the precondition assumes that P_1 holds, and that P_3 fulfils Q_1. The latter is formulated using a reactive weakest precondition (wp_r), which obeys standard laws [20] such as:

$$(\textstyle\bigvee_{i \in I} P(i))\; \mathit{wp}_r\; R = \bigwedge_{i \in I} P(i)\; \mathit{wp}_r\; R \qquad (P \fatsemi Q)\; \mathit{wp}_r\; R = P\; \mathit{wp}_r\; (Q\; \mathit{wp}_r\; R)$$

In the pericondition, either the first contract is intermediate (P_2), or else it terminated (P_3) and then following this the second is intermediate (Q_2). In the postcondition the contracts have both terminated in sequence ($P_3 \fatsemi Q_3$).

With these and related theorems [10], we can calculate contracts of reactive programs. Verification, then, can be performed by proving a refinement between two reactive contracts, a strategy we have mechanised in the Isabelle/UTP tactics rdes-refine and rdes-eq [10]. The question remains, though, how to reason about the underlying compositions of reactive relations for the pre-, peri-, and postconditions. For example, consider the action $(a \rightarrow \mathit{Skip}) \fatsemi x := v$. For its postcondition, we must simplify $(tt = \langle a \rangle \wedge st' = st) \fatsemi (st' = st \oplus \{x \mapsto v\} \wedge tt = \langle\rangle)$. In order to simplify its precondition, we also need to consider reactive weakest preconditions. Without such simplifications, reactive relations can grow very quickly and hamper proof. Finally, of particular importance is the handling of iterative reactive relations. We address these issues in this paper.

3 Linking UTP and Kleene Algebra

In this section, we characterise properties of a UTP theory sufficient to identify a KA, and use this to obtain theorems for iterative contracts. We observe that UTP relations form a KA $(Rel, \sqcap, ;, ^*, \mathbb{I})$, where $P^* \triangleq (\nu X \bullet \mathbb{I} \sqcap P ; X)$. We have proved this definition equivalent to the power form: $P^* = (\bigsqcap_{i \in \mathbb{N}} P^i)$ where P^n iterates sequential composition n times.

Typically, UTP theories, like $[\![\mathbf{NSRD}]\!]$, share the operators for choice ($\sqcap$) and composition ($;$), only redefining them when absolutely necessary. Formally, given a UTP theory defined by a healthiness condition $\mathbf{H}$, the set of healthy relations $[\![\mathbf{H}]\!]$ is closed under $\sqcap$ and $;$. This has the major advantage that a large body of laws is directly applicable from the relational calculus. The ubiquity of $\sqcap$, in particular, can be characterised through the subset of continuous UTP theories, where $\mathbf{H}$ distributes through arbitrary non-empty infima, that is,

$$\mathbf{H}\left(\bigsqcap_{i \in I} P(i)\right) = \bigsqcap_{i \in I} \mathbf{H}(P(i)) \text{ provided } I \neq \emptyset.$$

Monotonicity of $\mathbf{H}$ follows from continuity, and so such theories induce a complete lattice. Continuous UTP theories include designs [6,14], CSP, and *Circus* [7]. A further consequence of continuity is that the relational weakest fixed-point operator $\mu\, X \bullet F(X)$ constructs healthy relations when $F : Rel \to [\![\mathbf{H}]\!]$.

Though these theories share infima and weakest fixed-points, they do not, in general, share $\top$ and $\bot$ elements, which is why the infima are non-empty in the above continuity property. Rather, we have a top element $\top_{\mathbf{H}} \triangleq \mathbf{H}(\mathbf{false})$ and a bottom element $\bot_{\mathbf{H}} \triangleq \mathbf{H}(\mathbf{true})$ [3]. The theories also do not share the relational identity $\mathbb{I}$, but typically define a bespoke identity $\mathbb{I}_{\mathbf{H}}$, which means that $[\![\mathbf{H}]\!]$ is not closed under the relational Kleene star. However, $[\![\mathbf{H}]\!]$ is closed under the related Kleene plus $P^+ \triangleq P ; P^*$ since it is equivalent to $(\bigsqcap_{i \in \mathbb{N}} P^{i+1})$, which iterates P one or more times. Thus, we can obtain a theory Kleene star with the definition $P^{\mathbf{*}} \triangleq \mathbb{I}_{\mathbf{H}} \sqcap P^+$, under which $\mathbf{H}$ is indeed closed. We, therefore, define the following criteria for a UTP theory.

Definition 3.1. *A Kleene UTP theory $(\mathbf{H}, \mathbb{I}_{\mathbf{H}})$ satisfies the following conditions: (1) $\mathbf{H}$ is idempotent and continuous; (2) $\mathbf{H}$ is closed under sequential composition; (3) identity $\mathbb{I}_{\mathbf{H}}$ is $\mathbf{H}$-healthy; (4) $\mathbb{I}_{\mathbf{H}} ; P = P ; \mathbb{I}_{\mathbf{H}} = P$, when P is $\mathbf{H}$-healthy; (5) $\top_{\mathbf{H}} ; P = \top_{\mathbf{H}}$, when P is $\mathbf{H}$-healthy.*

From these properties, we can prove the following theorem.

Theorem 3.2. *If $(\mathbf{H}, \mathbb{I}_{\mathbf{H}})$ is a Kleene UTP theory, then $([\![\mathbf{H}]\!], \sqcap, \top_{\mathbf{H}}, ;, \mathbb{I}_{\mathbf{H}}, ^{\mathbf{*}})$ forms a weak Kleene algebra.*

Proof. We prove this in Isabelle/UTP by lifting of laws from the Isabelle/HOL KA hierarchy [21, 22]. For details see [11].

All the identities of Theorem 2.3 hold in a Kleene UTP theory, thus providing reasoning capabilities for iterative programs. In particular, we can show that $([\![\mathbf{NSRD}]\!], \sqcap, \mathbf{Miracle}, ;, \mathbb{I}_R, ^{\mathbf{*}})$ and $([\![\mathbf{RR}]\!], \sqcap, \mathbf{false}, ;, \mathbb{I}_r, ^{\mathbf{*}})$ both form weak KAs. Moreover, we can now also show how to calculate an iterative contract [12].

Theorem 3.3 (Reactive Contract Iteration).

$$[\, P \vdash Q \mid R \,]^* = [\, R^* \ \mathbf{wp}_r \ P \vdash R^* \fatsemi Q \mid R^* \,]$$

Note that the outer and inner star are different operators. The precondition states that R must not violate P after any number of iterations. The pericondition has R iterated followed by Q holding, since the final observation is intermediate. The postcondition simply iterates R. Thus we have the basis for calculating and reasoning about iterative contracts.

4 Reactive Relations of Stateful Failures-Divergences

Here, we specialise our contract theory to incorporate failure traces, which are used in CSP, *Circus*, and related languages [23]. We define atomic operators to describe the underlying reactive relations, and the associated equational theory to expand and simplify compositions arising from Theorems 2.6 and 3.3, and thus support automated reasoning. We consider external choice separately (§5).

Healthiness condition $\mathbf{NCSP} \triangleq \mathbf{NSRD} \circ \mathbf{CSP3} \circ \mathbf{CSP4}$ characterises the stateful failures-divergences model [6,7,9]. **CSP3** and **CSP4** ensure the refusal sets are well-formed [6,9]: ref' can only be mentioned in the pericondition (see also Appendix A). **NCSP**, like **NSRD**, is continuous and has **Skip**, defined below, as a left and right unit. Thus, $(\llbracket \mathbf{NCSP} \rrbracket, \sqcap, \mathbf{Miracle}, \fatsemi, \mathbf{Skip}, ^*)$ forms a Kleene algebra. An **NCSP** contract has the following specialised form [13].

$$[\, P(tt, st) \vdash Q(tt, st, ref') \mid R(tt, st, st') \,]$$

The underlying reactive relations capture a portion of the stateful failures-divergences. P captures the initial states and traces that do not induce divergence, that is, unpredictable behaviour like **Chaos**. Q captures the stateful failures of a program: the set of events not being refused (ref') having performed trace tt, starting in state st. R captures the terminated behaviours, where a final state is observed but no refusals. We describe the pattern of the underlying reactive relations using the following constructs.

Definition 4.1 (Reactive Relational Operators).

$$\mathcal{I}[b(st), t(st)] \triangleq \mathbf{RR}(b(st) \wedge t(st) \leq tt) \tag{1}$$

$$\mathcal{E}[b(st), t(st), E(st)] \triangleq \mathbf{RR}(b(st) \wedge tt = t(st) \wedge (\forall\, e \in E(st) \bullet e \notin ref')) \tag{2}$$

$$\Phi[b(st), \sigma, t(st)] \triangleq \mathbf{RR}(b(st) \wedge st' = \sigma(st) \wedge tt = t(st)) \tag{3}$$

In this definition, we utilise expressions b, t, and E that refer only to the variables by which they are parametrised. Namely, $b(st) : \mathbb{B}$ is a condition on st, $t(st) :$ seq *Event* is a trace expression that describes a possible event sequence in terms of st, and $E(st) : \mathbb{P}\, Event$ is an expression that describes a set of events. Following [24], we describe state updates with substitutions $\sigma : \Sigma \to \Sigma$. We use $(\!|x \mapsto v|\!)$ to denote a substitution, which is the identity for every variable, except that v is

assigned to x. Substitutions can also be applied to contracts and relations using operator $\sigma \dagger P$, and then $Q[v/x] \triangleq (\!|x \mapsto v|\!) \dagger Q$. This operator obeys similar laws to syntactic substitution, though it is a semantic operator [10].

$\mathcal{I}[b(\mathit{st}), t(\mathit{st})]$ is a specification of initial behaviour used in preconditions. It states that initially the state satisfies condition b, and t is a prefix of the overall trace. $\mathcal{E}[b(\mathit{st}), t(\mathit{st}), E(\mathit{st})]$ is used in pericondtions to specify quiescent observations, and corresponds to a failure trace. It specifies that the state variables initially satisfy b, the interaction described by t has occurred, and finally we reach a quiescent phase where none of the events in E are being refused. $\Phi[b(\mathit{st}), \sigma, t(\mathit{st})]$ is used to encode final terminated observations in the postcondition. It specifies that the initial state satisfies b, the state update σ is applied, and the interaction t has occurred.

These operators are all deterministic, in the sense that they describe a single interaction and state-update history. There is no need for explicit nondeterminism here, as this is achieved using $\bigvee$. These operators allow us to concisely specify the basic operators of our theory as given below.

Definition 4.2 (Basic Reactive Operators).

$$\langle \sigma \rangle_{\mathsf{c}} \triangleq [\, \mathbf{true}_r \vdash \mathbf{false} \mid \Phi[true, \sigma, \langle\rangle] \,] \tag{1}$$

$$\mathbf{Do}(a) \triangleq [\, \mathbf{true}_r \vdash \mathcal{E}[true, \langle\rangle, \{a\}] \mid \Phi[true, id, \langle a \rangle] \,] \tag{2}$$

$$\mathbf{Stop} \triangleq [\, \mathbf{true}_r \vdash \mathcal{E}[true, \langle\rangle, \emptyset] \mid \mathbf{false} \,] \tag{3}$$

Generalised assignment $\langle \sigma \rangle_{\mathsf{c}}$ is again inspired by [24]. It has a $\mathbf{true}_r$ precondition and a $\mathbf{false}$ pericondition: it has no intermediate observations. The postcondition states that for any initial state ($true$), the state is updated using σ, and no events are produced ($\langle\rangle$). A singleton assignment $x := v$ can be expressed using a state update $(\!|x \mapsto v|\!)$. We define $\mathbf{Skip} \triangleq \langle id \rangle_{\mathsf{c}}$, which leaves all variables unchanged.

$\mathbf{Do}(a)$ encodes an event action. Its pericondition states that no event has occurred, and a is accepted. Its postcondition extends the trace by a, leaving the state unchanged. We can denote *Circus* event prefix $a \rightarrow P$ as $\mathbf{Do}(a) \fatsemi P$.

Finally, $\mathbf{Stop}$ represents a deadlock: its pericondition states the trace is unchanged and no events are being accepted. The postcondition is false as there is no way to terminate. A *Circus* guard $g \mathbin{\&} P$ can be denoted as $(P \lhd g \rhd \mathbf{Stop})$, which behaves as P when g is true, and otherwise deadlocks.

To calculate contractual semantics, we need laws to reduce pre-, peri-, and postconditions. These need to cater for various composition cases of operators $\bigsqcap$, $\fatsemi$, and $\Box$. So, we prove [13] the following composition laws for $\mathcal{E}$ and Φ.

Theorem 4.3 (Reactive Relational Compositions).

$$[b]^{\top}_{r} \fatsemi P = \Phi[b, id, \langle\rangle] \fatsemi P \tag{1}$$

$$\Phi[b_1, \sigma_1, t_1] \fatsemi \Phi[b_2, \sigma_2, t_2] = \Phi[b_1 \wedge \sigma_1 \dagger b_2, \sigma_2 \circ \sigma_1, t_1 \frown \sigma_1 \dagger t_2] \tag{2}$$

$$\Phi[b_1, \sigma_1, t_1] \fatsemi \mathcal{E}[b_2, t_2, E] = \mathcal{E}[b_1 \wedge \sigma_1 \dagger b_2, t_1 \frown \sigma_1 \dagger t_2, \sigma_1 \dagger E] \tag{3}$$

$$\Phi[b_1, \sigma_1, t_1] \lhd c \rhd \Phi[b_2, \sigma_2, t_2] = \Phi[b_1 \lhd c \rhd b_2, \sigma_1 \lhd c \rhd \sigma_2, t_1 \lhd c \rhd t_2] \tag{4}$$

$$\mathcal{E}[b_1, t_1, E_1] \lhd c \rhd \mathcal{E}[b_2, t_2, E_2] = \mathcal{E}[b_1 \lhd c \rhd b_2, t_1 \lhd c \rhd t_2, E_1 \lhd c \rhd E_2] \quad (5)$$

$$\left(\bigwedge_{i \in I} \mathcal{E}[b(i), t, E(i)]\right) = \mathcal{E}\left[\bigwedge_{i \in I} b(i), t, \bigcup_{i \in I} E(i)\right] \quad (6)$$

Law (1) states that a precomposed test can be expressed using Φ. (2) states that the composition of two terminated observations results in the conjunction of the state conditions, composition of the state updates, and concatenation of the traces. It is necessary to apply the initial state update σ_1 as a substitution to both the second state condition (s_2) and the trace expression (t_2). (3) is similar, but accounts for the enabled events rather than state updates. (2) and (3) are required because of Theorem 2.6-2, which sequentially composes a pericondition with a postcondition, and a postcondition with a postcondition. (4) and (5) show how conditional distributes through the operators. Finally, (6) shows that a conjunction of intermediate observations with a common trace takes the conjunction of the state conditions, and the union of the enabled events. It is needed for external choice, which conjoins the periconditions (see §5).

In order to calculate preconditions, we need to consider the weakest precondition operator. Theorem 2.6-2 requires that, in a sequential composition $P \fatsemi Q$, we need to show that the postcondition of contract P satisfies the precondition of contract Q. Theorem 4.3 explains how to eliminate most composition operators in a contract's postcondition, but not in general $\vee$. Postconditions are, therefore, typically expressed as disjunctions of the Φ operator, and so it suffices to calculate its weakest precondition using the theorem below.

Theorem 4.4. $\Phi[s, \sigma, t]$ $\mathbf{wp}_r$ $P = (\mathcal{I}[s, t] \Rightarrow (\sigma \dagger P)[\mathit{tt} - t/\mathit{tt}])$

In order for $\Phi[s, \sigma, t]$ to satisfy reactive condition P, whenever we start in the state satisfying s and the trace t has been performed, P must hold on the remainder of the trace ($\mathit{tt} - t$), and with the state update σ applied. We can now use these laws, along with Theorem 2.6, to calculate the semantics of processes, and to prove equality and refinement conjectures, as we illustrate below.

Example 4.5. We show that $(x{:=}1 \fatsemi \mathbf{Do}(a.x) \fatsemi x := x{+}2) = (\mathbf{Do}(a.1) \fatsemi x{:=}3)$. By applying Definition 4.2 and Theorems 2.6 (2), 4.3, 4.4, both sides reduce to $[\,\mathbf{true}_r \vdash \mathcal{E}[\mathit{true}, \langle\rangle, \{a.1\}] \mid \Phi[\mathit{true}, \{x \mapsto 3\}, \langle a.1\rangle]\,]$, which has a single quiescent state, waiting for event $a.1$, and a single final state, where $a.1$ has occurred and state variable x has been updated to 3. We calculate the left-hand side below.

$$
\begin{aligned}
&(x{:=}1 \fatsemi \mathbf{Do}(a.x) \fatsemi x := x + 2) \\
&= \left(\begin{array}{l} [\,\mathbf{true}_r \vdash \mathbf{false} \mid \Phi[\mathit{true}, (\!|x \mapsto 1|\!), \langle\rangle]\,] \fatsemi \\ [\,\mathbf{true}_r \vdash \mathcal{E}[\mathit{true}, \langle\rangle, \{a.x\}] \mid \Phi[\mathit{true}, \mathit{id}, \langle a.x\rangle]\,] \fatsemi \\ [\,\mathbf{true}_r \vdash \mathbf{false} \mid \Phi[\mathit{true}, (\!|x \mapsto x + 1|\!), \langle\rangle]\,] \end{array}\right) && [\text{Def. 4.2}] \\
&= \left[\, \mathbf{true}_r \vdash \begin{array}{l} \Phi[\mathit{true}, (\!|x \mapsto 1|\!), \langle\rangle] \fatsemi \\ \mathcal{E}[\mathit{true}, \langle\rangle, \{a.x\}] \end{array} \;\middle|\; \begin{array}{l} \Phi[\mathit{true}, (\!|x \mapsto 1|\!), \langle\rangle] \fatsemi \\ \Phi[\mathit{true}, \mathit{id}, \langle a.x\rangle] \fatsemi \\ \Phi[\mathit{true}, (\!|x \mapsto x + 2|\!), \langle\rangle] \end{array} \,\right] && \left[\begin{array}{l}\text{Thm. 2.6,}\\ \text{Thm. 4.4}\end{array}\right] \\
&= \left[\, \mathbf{true}_r \vdash \mathcal{E}[\mathit{true}, \langle\rangle[1/x], \{a.x\}[1/x]] \;\middle|\; \begin{array}{l} \Phi[\mathit{true}, (\!|x \mapsto 1|\!), \langle a.1\rangle] \fatsemi \\ \Phi[\mathit{true}, (\!|x \mapsto x{+}2|\!), \langle\rangle] \end{array} \,\right] && [\text{Thm. 4.3}]
\end{aligned}
$$

$= [\, \textbf{\textit{true}}_r \vdash \mathcal{E}[true, \langle\rangle, \{a.1\}] \mid \Phi[true, \{x \mapsto 3\}, \langle a.1\rangle]\,]$ □

Similarly, we can use our theorems, with the help of our mechanised proof strategy in rdes-eq, to prove a number of general laws [13].

Theorem 4.6 (Stateful Failures-Divergences Laws).

$$\langle\sigma\rangle_{\mathsf{c}} \fatsemi [P_1 \vdash P_2 \mid P_3] = [\sigma\dagger P_1 \vdash \sigma\dagger P_2 \mid \sigma\dagger P_3] \quad (1) \qquad \langle\sigma\rangle_{\mathsf{c}} \fatsemi \langle\rho\rangle_{\mathsf{c}} = \langle\rho \circ \sigma\rangle_{\mathsf{c}} \quad (3)$$

$$\langle\sigma\rangle_{\mathsf{c}} \fatsemi \textbf{\textit{Do}}(e) = \textbf{\textit{Do}}(\sigma \dagger e) \fatsemi \langle\sigma\rangle_{\mathsf{c}} \quad (2) \qquad \textbf{\textit{Stop}} \fatsemi P = \textbf{\textit{Stop}} \quad (4)$$

Law (1) shows how assignment distributes substitutions through a contract. (2) and (3) are consequences of (1). (4) shows that deadlock is a left annihilator.

5 External Choice and Productivity

In this section we consider reasoning about programs with external choice, and characterise the important subclass of productive contracts [3], which are also essential in verifying recursive and iterative reactive programs.

An external choice $P \,\Box\, Q$ resolves whenever either P or Q engages in an event or terminates. Thus, its semantics requires that we filter observations with a non-empty trace. We introduce healthiness condition $\textbf{\textit{R4}}(P) \triangleq (P \wedge \mathit{tt} > \langle\rangle)$, whose fixed points strictly increase the trace, and its dual $\textbf{\textit{R5}}(P) \triangleq (P \wedge \mathit{tt} = \langle\rangle)$ where the trace is unchanged. We use these to define indexed external choice.

Definition 5.1 (Indexed External Choice).

$$\Box\, i \in I \bullet [\, P_1(i) \vdash P_2(i) \mid P_3(i)\,] \triangleq$$
$$\left[\, \bigwedge_{i\in I} P_1(i) \vdash \left(\bigwedge_{i\in I} \textbf{\textit{R5}}(P_2(i))\right) \vee \left(\bigvee_{i\in I} \textbf{\textit{R4}}(P_2(i))\right) \mid \bigvee_{i\in I} P_3(i)\,\right]$$

This enhances the binary definition [6, 7], and recasts our definition in [3] for calculation. Like nondeterministic choice, the precondition requires that all constituent preconditions are satisfied. In the pericondition ***R4*** and ***R5*** filter all observations. We take the conjunction of all ***R5*** behaviours: no event has occurred, and all branches are offering to communicate. We take the disjunction of all ***R4*** behaviours: an event occurred, and the choice is resolved. In the postcondition the choice is resolved, either by communication or termination, and so we take the disjunction of all constituent postconditions. Since unbounded choice is supported, we can denote indexed input prefix for any size of input domain A:

$$a?x : A \to P(x) \;\triangleq\; \Box\, x \in A \bullet a.x \to P(x)$$

We next show how ***R4*** and ***R5*** filter the various reactive relational operators, which can be applied to reason about contracts involving external choice.

Theorem 5.2 (Trace Filtering).

$$\textbf{\textit{R4}}\left(\bigvee_{i\in I} P(i)\right) = \bigvee_{i\in I} \textbf{\textit{R4}}(P(i)) \qquad \textbf{\textit{R5}}\left(\bigvee_{i\in I} P(i)\right) = \bigvee_{i\in I} \textbf{\textit{R5}}(P(i))$$

$$\textbf{\textit{R4}}(\Phi[s, \sigma, \langle\rangle]) = \textbf{\textit{false}} \qquad \textbf{\textit{R5}}(\mathcal{E}[s, \langle\rangle, E]) = \mathcal{E}[s, \langle\rangle, E]$$

$$\textbf{\textit{R4}}(\Phi[s, \sigma, \langle a, ...\rangle]) = \Phi[s, \sigma, \langle a, ...\rangle] \qquad \textbf{\textit{R5}}(\mathcal{E}[s, \langle a, ...\rangle, E]) = \textbf{\textit{false}}$$

Both operators distribute through $\bigvee$. Relations that produce an empty trace yield ***false*** under ***R4*** and are unchanged under ***R5***. Relations that produce a non-empty trace yield ***false*** for ***R5***, and are unchanged underr ***R4***. We can now filter the behaviours that do and do not resolve the choice, as exemplified below.

Example 5.3. Consider the calculation of $a \to b \to \textbf{\textit{Skip}} \;\Box\; c \to \textbf{\textit{Skip}}$. The left branch has two quiescent observations, one waiting for a, and one for b having performed a: its pericondition is $\mathcal{E}[true, \langle\rangle, \{a\}] \vee \mathcal{E}[true, \langle a\rangle, \{b\}]$. Application of ***R5*** to this will yield the first disjunct, since the trace has not increased, and ***R4*** will yield the second disjunct. For the right branch there is one quiescent observation, $\mathcal{E}[true, \langle\rangle, \{c\}]$, which contributes an empty trace and is ***R5*** only. The overall pericondition is $(\mathcal{E}[true, \langle\rangle, \{a\}] \wedge \mathcal{E}[true, \langle\rangle, \{c\}]) \vee \mathcal{E}[true, \langle a\rangle, \{b\}]$, which is simply $\mathcal{E}[true, \langle\rangle, \{a, c\}] \vee \mathcal{E}[true, \langle a\rangle, \{b\}]$. □

By calculation we can now prove that $(\llbracket \textbf{\textit{NCSP}} \rrbracket, \Box, \textbf{\textit{Stop}})$ forms a commutative and idempotent monoid, and ***Chaos***, the divergent program, is its annihilator. Sequential composition also distributes from the left and right through external choice, but only when the choice branches are productive [3].

Definition 5.4. *A contract* $[\, P_1 \vdash P_2 \mid P_3 \,]$ *is productive when* P_3 *is* ***R4*** *healthy.*

A productive contract is one that, whenever it terminates, strictly increases the trace. For example $a \to \textbf{\textit{Skip}}$ is productive, but ***Skip*** is not. Constructs that do not terminate, like ***Chaos***, are also productive. The imposition of ***R4*** ensures that only final observations that increase the trace, or are ***false***, are admitted.

We define healthiness condition ***PCSP***, which extends ***NCSP*** with productivity. We also define ***ICSP***, which formalises instantaneous contracts where the postcondition is ***R5*** healthy and the pericondition is ***false***. For example, both ***Skip*** and $x := v$ are ***ICSP*** healthy as they do not contribute to the trace and have no intermediate observations. This allows us to prove the following laws.

Theorem 5.5 (External Choice Distributivity).

$$(\Box\, i \in I \bullet P(i)) \fatsemi Q \;=\; \Box\, i \in I \bullet (P(i) \fatsemi Q) \qquad [\textit{if, } \forall\, i \in I, P(i) \textit{ is } \textbf{\textit{PCSP}} \textit{ healthy}]$$

$$P \fatsemi (\Box\, i \in I \bullet Q(i)) \;=\; \Box\, i \in I \bullet (P \fatsemi Q(i)) \qquad [\textit{if } P \textit{ is } \textbf{\textit{ICSP}} \textit{ healthy}]$$

The first law follows because every $P(i)$, being productive, must resolve the choice before terminating, and thus it is not possible to reach Q before this occurs. It generalises the standard guarded choice distribution law for CSP [6, page 211]. The second law follows for the converse reason: since P cannot resolve the choice with any of its behaviour, it is safe to execute it first.

Productivity also forms an important criterion for guarded recursion that we utilise in §6 to calculate fixed points. ***PCSP*** is closed under several operators.

Theorem 5.6. (1) ***Miracle***, ***Stop***, ***Do***(a) *are* ***PCSP****;* (2) $P \fatsemi Q$ *is* ***PCSP*** *if either* P *or* Q *is* ***PCSP****;* (3) $\Box\, i \in I \bullet P(i)$ *is* ***PCSP*** *if, for all* $i \in I$, $P(i)$ *is* ***PCSP****.*

Calculation of external choice is now supported, and a notion of productivity defined. In the next section we use the latter for calculation of while-loops.

6 While Loops and Reactive Invariants

In this section we complete our verification strategy by adding support for iteration. Iterative programs can be constructed using the reactive while loop.

$$b \circledast P \triangleq (\mu\, X \bullet P \fatsemi X \lhd b \rhd \textbf{\textit{Skip}}).$$

We use the weakest fixed-point so that an infinite loop with no observable activity corresponds to the divergent action ***Chaos***, rather than ***Miracle***. For example, we can show that $(\mathit{true} \circledast\ x := x+1\,) = \textbf{\textit{Chaos}}$. The *true* condition is not a problem in this context because, unlike its imperative cousin, the reactive while loop pauses for interaction with its environment during execution, and therefore infinite executions are observable and therefore potentially useful.

In order to reason about such behaviour, we need additional calculational laws. A fixed-point $(\mu\, X \bullet F(X))$ is guarded provided at least one event is contributed to the trace by F prior to it reaching X. For instance, $\mu\, X \bullet a \rightarrow X$ is guarded, but $\mu\, X \bullet y := 1 \fatsemi X$ is not. Hoare and He's theorem [6, theorem 8.1.13, page 206] states that if F is guarded, then there is a unique fixed-point and hence $(\mu\, X \bullet F(X)) = (\nu X \bullet F(X))$. Then, provided F is continuous, we can invoke Kleene's fixed-point theorem to calculate νF. Our previous result [3] shows that if P is productive, then $\lambda\, X \bullet P \fatsemi X$ is guarded, and so we can calculate its fixed-point. We now generalise this for the function above.

Theorem 6.1. *If P is productive, then $(\mu\, X \bullet P \fatsemi X \lhd b \rhd \textbf{\textit{Skip}})$ is guarded.*

Proof. In addition to our previous theorem [3], we use the following properties:

- If X is not mentioned in P then $\lambda\, X \bullet P$ is guarded;
- If F and G are both guarded, then $\lambda\, X \bullet F(X) \lhd b \rhd G(X)$ is guarded. □

This allows us to convert the fixed-point into an iterative form. In particular, we can prove the following theorem that expresses it in terms of Kleene star.

Theorem 6.2. *If P is **PCSP** healthy then $b \circledast P = ([b]^{\top}_r \fatsemi P)^* \fatsemi [\neg b]^{\top}_r$.*

This theorem is similar to the usual imperative definition [21,22]. P is executed multiple times when b is true initially, but each run concludes when b is false. However, due to the embedding of reactive behaviour, there is more going on than meets the eye; the next theorem shows how to calculate an iterative contract.

Theorem 6.3. *If R is **R4** healthy then*

$$b \circledast [P \vdash Q \mid R] = [\,([b]^{\top}_r \fatsemi R)^*\ \textbf{\textit{wp}}_r\,(b \Rightarrow P) \vdash ([b]^{\top}_r \fatsemi R)^* \fatsemi [b]^{\top}_r \fatsemi Q \mid ([b]^{\top}_r \fatsemi R)^* \fatsemi [\neg b]^{\top}_r\,]$$

The precondition requires that any number of R iterations, where b is initially true, satisfies P. This ensures that the contract does not violate its own precondition from one iteration to the next. The pericondition states that intermediate observations have R executing several times, with b true, and following this b remains true and the contract is quiescent (Q). The postcondition is similar, but

after several iterations, b becomes false and the loop terminates, which is the standard relational form of a while loop.

Theorem 6.3 can be utilised to prove a refinement introduction law for the reactive while loop. This employs "reactive invariant" relations, which describe how both the trace and state variables are permitted to evolve.

Theorem 6.4. $[I_1 \vdash I_2 \mid I_3] \sqsubseteq b \circledast [Q_1 \vdash Q_2 \mid Q_3]$ *provided that:*

1. *the assumption is weakened* $(([b]^\top_r \fatsemi Q_3)^\star \ \mathbf{wp}_r\, (b \Rightarrow Q_1) \sqsubseteq I_1)$*;*
2. *when* b *holds,* Q_2 *establishes the* I_2 *pericondition invariant* $(I_2 \sqsubseteq ([b]^\top_r \fatsemi Q_2))$ *and,* Q_3 *maintains it* $(I_2 \sqsubseteq [b]^\top_r \fatsemi Q_3 \fatsemi I_2)$*;*
3. *postcondition invariant* I_3 *is established when* b *is false* $(I_3 \sqsubseteq [\neg b]^\top_r)$ *and* Q_3 *establishes it when* b *is true* $(I_3 \sqsubseteq [b]^\top_r \fatsemi Q_3 \fatsemi I_3)$*.*

Proof. By application of refinement introduction, with Theorems 2.2-3 and 6.3.

Theorem 6.4 shows the conditions under which an iterated reactive contract satisfies an invariant contract $[I_1 \vdash I_2 \mid I_3]$. Relations I_2 and I_3 are reactive invariants that must hold in quiescent and final observations, respectively. Both can refer to st and tt, I_2 can additionally refer to ref', and I_3 to st'. Combined with the results from §4 and §5, this result provides the basis for a proof strategy for iterative reactive programs that we now exemplify.

7 Verification Strategy for Reactive Programs

Our collected results give rise to an automated verification strategy for iterative reactive programs, whereby we (1) calculate the contract of a reactive program, (2) use our equational theory to simplify the underlying reactive relations, (3) identify suitable invariants for reactive while loops, and (4) finally prove refinements using relational calculus. Although the underlying relations can be quite complex, our equational theory from §4 and §5, aided by the Isabelle/HOL simplifier, can be used to rapidly reduce them to more compact forms amenable to automated proof. In this section we illustrate this strategy with the buffer in Example 2.5. We prove two properties: (1) deadlock freedom, and (2) that the order of values produced is the same as those consumed.

We first calculate the contract of the main loop in the *Buffer* process and then use this to calculate the overall contract for the iterative behaviour.

Theorem 7.1 (Loop Body). *The body of the loop is* $[\, \mathbf{true}_r \vdash B_2 \mid B_3 \,]$ *where*

$$B_2 = \mathcal{E}\big[\mathit{true}, \langle\rangle, \textstyle\bigcup_{v \in \mathbb{N}} \{\mathit{inp}.v\} \cup (\{\mathit{out}.\mathit{head}(\mathit{bf})\} \lhd 0 < \#\mathit{bf} \rhd \emptyset)\big]$$

$$B_3 = \begin{pmatrix} (\bigvee_{v \in \mathbb{N}} \Phi[\mathit{true}, \{\mathit{bf} \mapsto \mathit{bf} \frown \langle v \rangle\}, \langle \mathit{inp}.v \rangle]) \vee \\ \Phi[0 < \#\mathit{bf}, \{\mathit{bf} \mapsto \mathit{tail}(\mathit{bf})\}, \langle \mathit{out}.\mathit{head}(\mathit{bf}) \rangle] \end{pmatrix}$$

The $\mathbf{true}_r$ precondition implies no divergence. The pericondition states that every input event is enabled, and the output event is enabled if the buffer is non-empty. The postcondition contains two possible final observations: (1) an input

event occurred and the buffer variable was extended; or (2) provided the buffer was non-empty initially, then the buffer's head is output and bf is contracted.

Proof. To exemplify, we calculate the left-hand side of the choice, employing Theorems 2.6, 4.2, 4.3, and 5.2. The entire calculation is automated in Isabelle/UTP.

$$
\begin{aligned}
& inp?v \rightarrow bf := bf \frown \langle v \rangle \\
&= \Box\, v \in \mathbb{N} \bullet \mathbf{Do}(inp.v) \fatsemi bf := bf \frown \langle v \rangle && \text{[Defs]} \\
&= \Box\, v \in \mathbb{N} \bullet \begin{pmatrix} [\, \mathbf{true}_r \vdash \mathcal{E}[true, \langle\rangle, \{inp.v\}] \mid \Phi[true, id, \langle inp.v \rangle]\,] \fatsemi \\ [\, \mathbf{true}_r \vdash \mathbf{false} \mid \Phi[true, (\!| bf \mapsto bf \frown \langle v \rangle |\!), \langle\rangle]\,] \end{pmatrix} && [4.2] \\
&= \Box\, v \in \mathbb{N} \bullet \left[\mathbf{true}_r \vdash \begin{array}{l|l} \mathcal{E}[true, \langle\rangle, \{inp.v\}] & \Phi[true, id, \langle inp.v \rangle] \fatsemi \\ \vee\ \mathbf{false} & \Phi[true, (\!| bf \mapsto bf \frown \langle v \rangle |\!), \langle\rangle] \end{array} \right] && [2.6, 4.4] \\
&= \Box\, v \in \mathbb{N} \bullet [\mathbf{true}_r \vdash \mathcal{E}[true, \langle\rangle, \{inp.v\}] \mid \Phi[true, (\!| bf \mapsto bf \frown \langle v \rangle |\!), \langle inp.v \rangle]] && [4.3] \\
&= \left[\mathbf{true}_r \vdash \mathcal{E}\left[true, \langle\rangle, \bigcup_{v \in \mathbb{N}} \{inp.v\} \right] \,\middle|\, \bigvee_{v \in \mathbb{N}} \Phi[true, (\!| bf \mapsto bf \frown \langle v \rangle |\!), \langle inp.v \rangle] \right] && [5.1, 5.2]
\end{aligned}
$$

Though this calculation seems complicated, in practice it is fully automated and thus a user need not be concerned with these minute calculational details, but can rather focus on finding suitable reactive invariants. □

Then, by Theorem 6.3 we can calculate the overall behaviour of the buffer.

$$Buffer = [\, \mathbf{true}_r \vdash \Phi[true, \{bf \mapsto \langle\rangle\}, \langle\rangle] \fatsemi B_3^* \fatsemi B_2 \mid \mathbf{false}\,]$$

This is a non-terminating contract where every quiescent behaviour begins with an empty buffer, performs some sequence of buffer inputs and outputs accompanied by state updates (B_3^*), and is finally offering the relevant input and output events (B_2). We can now employ Theorem 6.4 to verify the buffer. First, we tackle deadlock freedom, which can be proved using the following refinement.

Theorem 7.2 (Deadlock Freedom).

$$\left[\mathbf{true}_r \,\middle|\, \bigvee_{s,t,E,e} \mathcal{E}[s, t, \{e\} \cup E] \,\middle|\, \mathbf{true}_r \right] \sqsubseteq Buffer$$

Since only quiescent observations can deadlock, we only constrain the pericondition. It characterises observations where at least one event e is being accepted: there is no deadlock. This theorem can be discharged automatically in 1.8s on an Intel i7-4790 desktop machine. We next tackle the second property.

Theorem 7.3 (Buffer Order Property). *The sequence of items output is a prefix of those that were previously input. This can be formally expressed as*

$$[\, \mathbf{true}_r \vdash outps(tt) \leq inps(tt) \mid \mathbf{true}_r\,] \sqsubseteq Buffer$$

where $inps(t), outps(t) : \mathrm{seq}\ \mathbb{N}$ extract the sequence of input and output elements from the trace t, respectively. The postcondition is left unconstrained as Buffer does not terminate.

Proof. First, we identify the reactive invariant $I \triangleq outps(tt) \leq bf \frown inps(tt)$, and show that $[\mathbf{true}_r \vdash I \mid \mathbf{true}_r] \sqsubseteq \mathbf{true}_r \circledast [\mathbf{true}_r \vdash B_2 \mid B_3]$. By Theorem 6.4 it suffices to show case (2), that is $I \sqsubseteq B_2$ and $I \sqsubseteq B_3 \fatsemi I$, as the other two cases are vacuous. These two properties can be discharged by relational calculus. Second, we prove that $[\mathbf{true}_r \vdash outps(tt) \leq inps(tt) \mid \mathbf{true}_r] \sqsubseteq bf := \langle\rangle \fatsemi [\mathbf{true}_r \vdash I \mid \mathbf{true}_r]$. This holds, by Theorem 4.6-1, since $I[\langle\rangle/bf] = outps(tt) \leq inps(tt)$. Thus, the overall theorem holds by monotonicity of $\fatsemi$ and transitivity of $\sqsubseteq$. The proof is semi-automatic — since we have to manually apply induction with Theorem 6.4 — with the individual proof steps taking 2.2s in total. □

8 Conclusion

We have demonstrated an effective verification strategy for reactive programs employing reactive relations and Kleene algebra. Our theorems and verification tool can be found in our theory repository[2], together with companion proofs.

Related work includes the works of Struth *et al.* on verification of imperative programs [21,22] using Kleene algebra for verification-condition generation, which our work heavily draws upon. Automated proof support for the failures-divergences model was previously provided by the CSP-Prover tool [25], which can be used to verify infinite-state systems in CSP. Our work is different both in its contractual semantics, and also in our explicit handling of state, which allows us to express variable assignments.

Our work also lies within the "design-by-contract" field [4]. The refinement calculus of reactive systems [26] is a language based on property transformers containing trace information. Like our work, they support reactive systems that are non-deterministic, non-input-receptive, and infinite state. The main differences are our handling of state variables, the basis in relational calculus, and our failures-divergences semantics. Nevertheless, our contract framework [3] can be linked to those results, and we expect to derive an assume-guarantee calculus.

In future work, we will extend our calculation strategy to parallel composition. We aim to apply it to more substantial examples, and are currently using it to build a prototype tactic for verifying robotic controllers [27]. In this direction, our semantics and techniques will be also be extended to cater for real-time, probabilistic, and hybrid computational behaviours [19].

Acknowledgments

This research is funded by the RoboCalc project[3], EPSRC grant EP/M025756/1.

References

1. Harel, D., Pneuli, A.: On the development of reactive systems. In: Logics and Models of Concurrent Systems. Volume 13 of NATO ASI. Springer (1985)

[2] Isabelle/UTP: https://github.com/isabelle-utp/utp-main
[3] RoboCalc Project: https://www.cs.york.ac.uk/circus/RoboCalc/

2. Bainomugisha, E., Carreton, A.L., Cutsem, T.V., Mostinckx, S., De Meuter, W.: A survey on reactive programming. ACM Computing Surveys **45**(4) (August 2013)
3. Foster, S., Cavalcanti, A., Canham, S., Woodcock, J., Zeyda, F.: Unifying theories of reactive design contracts. Submitted to Theoretical Computer Science (Dec 2017) Preprint: https://arxiv.org/abs/1712.10233.
4. Meyer, B.: Applying "design by contract". IEEE Computer **25**(10) (1992) 40–51
5. Hehner, E.C.R.: A Practical Theory of Programming. Springer (1993)
6. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice-Hall (1998)
7. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP semantics for Circus. Formal Aspects of Computing **21** (2009) 3–32
8. Kozen, D.: On Kleene algebras and closed semirings. In: MFCS. Volume 452 of LNCS., Springer (1990) 26–47
9. Cavalcanti, A., Woodcock, J.: A tutorial introduction to CSP in unifying theories of programming. In: PSSE. Volume 3167 of LNCS. Springer (2006) 220–268
10. Foster, S., Zeyda, F., Woodcock, J.: Unifying heterogeneous state-spaces with lenses. In: ICTAC. LNCS 9965, Springer (2016)
11. Foster, S.: Kleene algebra in Unifying Theories of Programming. Technical report, University of York (2018) http://eprints.whiterose.ac.uk/129359/.
12. Foster, S., et al.: Reactive designs in Isabelle/UTP. Technical report, University of York (2018) http://eprints.whiterose.ac.uk/129386/.
13. Foster, S., et al.: Stateful-failure reactive designs in Isabelle/UTP. Technical report, University of York (2018) http://eprints.whiterose.ac.uk/129768/.
14. Guttman, W., Möller, B.: Normal design algebra. Journal of Logic and Algebraic Programming **79**(2) (February 2010) 144–173
15. Möller, B., Höfner, P., Struth, G.: Quantales and temporal logics. In: AMAST. Volume 4019 of LNCS., Springer (2006) 263–277
16. Santos, T., Cavalcanti, A., Sampaio, A.: Object-Orientation in the UTP. In: UTP 2006. Volume 4010 of LNCS., Springer (2006) 20–38
17. Sherif, A., Cavalcanti, A., He, J., Sampaio, A.: A process algebraic framework for specification and validation of real-time systems. Formal Aspects of Computing **22**(2) (2010) 153–191
18. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
19. Foster, S., Cavalcanti, A., Woodcock, J., Zeyda, F.: Unifying theories of time with generalised reactive processes. Information Processing Letters **135** (2018) 47–52
20. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Communications of the ACM **18**(8) (1975) 453–457
21. Armstrong, A., Gomes, V., Struth, G.: Building program construction and verification tools from algebraic principles. Formal Aspects of Computing **28**(2) (2015)
22. Gomes, V.B.F., Struth, G.: Modal Kleene algebra applied to program correctness. In: Formal Methods. Volume 9995 of LNCS., Springer (2016) 310–325
23. Zhan, N., Kang, E.Y., Liu, Z.: Component publications and compositions. In: UTP. Volume 5713 of LNCS., Springer (2008) 238–257
24. Back, R.J., Wright, J.: Refinement Calculus: A Systematic Introduction. Springer (1998)
25. Isobe, Y., Roggenbach, M.: CSP-Prover: a proof tool for the verification of scalable concurrent systems. Journal of Computer Software, Japan Society for Software Science and Technology **25**(4) (2008) 85–92
26. Preoteasa, V., Dragomir, I., Tripakis, S.: Refinement calculus of reactive systems. In: Intl. Conf. on Embedded Systems (EMSOFT), IEEE (October 2014)
27. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J.: Automatic property checking of robotic applications. In: Intl. Conf. on Intelligent Robots and Systems (IROS), IEEE (2017) 3869–3876

A UTP Theory Definitions

In this appendix, we summarise our theory of reactive design contracts. The definitions are all mechanised in accompanying Isabelle/HOL reports [12,13].

A.1 Observational Variables

We declare two sets $\mathcal{T}$ and Σ that denote the sets of traces and state spaces, respectively, and operators $\frown : \mathcal{T} \to \mathcal{T} \to \mathcal{T}$ and $\varepsilon : \mathcal{T}$. We require that $(\mathcal{T}, \frown, \varepsilon)$ forms a trace algebra [19], which is a form of cancellative monoid. Example models include $(\mathbb{N}, +, 0)$ and $(\text{seq } A, \frown, \langle\rangle)$. We declare the following observational variables that are used in both our UTP theories:

$ok, ok' : \mathbb{B}$ – indicate divergence in the prior and present relation;
$wait, wait' : \mathbb{B}$ – indicate quiescence in the prior and present relation;
$\mathit{st}, \mathit{st}' : \Sigma$ – the initial and final state;
$tr, tr' : \mathcal{T}$ – the trace of the prior and present relation.

Since the theory is extensible, we also allow further observational variables to be added, which are denoted by the variables r and r'.

A.2 Healthiness Conditions

We first describe the healthiness conditions of reactive relations.

Definition A.1 (Reactive Relation Healthiness Conditions).

$$\begin{aligned} \mathbf{R1}(P) &\triangleq P \wedge tr \leq tr' \\ \mathbf{R2}_c(P) &\triangleq P[\varepsilon, tr' - tr/tr, tr'] \lhd tr \leq tr' \rhd P \\ \mathbf{RR}(P) &\triangleq \exists(ok, ok', wait, wait') \bullet \mathbf{R1}(\mathbf{R2}_c(P)) \end{aligned}$$

RR healthy relations do not refer to ok or $wait$ and have a well-formed trace associated with them. The latter is ensured by the reactive process healthiness conditions [6,9,19], **R1** and $\mathbf{R2}_c$, which justify the existence of the trace pseudo variable $\mathit{tt} \triangleq tr' - tr$. **RR** is closed under relational calculus operators **false**, $\vee$, $\wedge$, and $;$, but not **true**, $\neg$, $\Rightarrow$, or $\mathbb{I}$. We therefore define healthy versions below.

Definition A.2 (Reactive Relation Operators).

$$\begin{aligned} \mathbf{true}_r &\triangleq \mathbf{R1}(true) \\ \neg_r P &\triangleq \mathbf{R1}(\neg P) \\ P \Rightarrow_r Q &\triangleq \neg_r P \vee Q \end{aligned} \qquad \begin{aligned} P \ \mathbf{wp}_r \ Q &\triangleq \neg_r (P ; (\neg_r Q)) \\ \mathbb{I}_r &\triangleq (tr' = tr \wedge \mathit{st}' = \mathit{st} \wedge r' = r) \end{aligned}$$

We define a reactive complement $\neg_r P$, reactive implication $P \Rightarrow_r Q$, and reactive true $\mathbf{true}_r$, which with the other connectives give rise to a Boolean algebra [3]. We also define the reactive skip $\mathbb{I}_r$, which is the unit of $;$, and the reactive weakest precondition operator $\mathbf{wp}_r$. The latter is similar to the standard UTP definition of weakest precondition [6], but uses the reactive complement.

We next define the healthiness conditions of reactive contracts.

Definition A.3 (Reactive Designs Healthiness Conditions).

$$\begin{array}{rcl@{\qquad}rcl}
\mathbf{R3}_h(P) & \triangleq & \mathbb{I}_R \lhd wait \rhd P & \mathbb{I}_R & \triangleq & \mathbf{RD1}((\exists\, st \bullet \mathbb{I}_r) \lhd wait \rhd \mathbb{I}_r) \\
\mathbf{RD1}(P) & \triangleq & ok \Rightarrow_r P & \mathbf{R}_s & \triangleq & \mathbf{R1} \circ \mathbf{R2}_c \circ \mathbf{R3}_h \\
\mathbf{RD2}(P) & \triangleq & P \fatsemi \mathbf{J} & \mathbf{SRD}(P) & \triangleq & \mathbf{RD1} \circ \mathbf{RD2} \circ \mathbf{R}_s \\
\mathbf{RD3}(P) & \triangleq & P \fatsemi \mathbb{I}_R & \mathbf{NSRD}(P) & \triangleq & \mathbf{RD1} \circ \mathbf{RD3} \circ \mathbf{R}_s
\end{array}$$

$\mathbf{R3}_h$ states that if the predecessor is waiting then a reactive design behaves like $\mathbb{I}_R$, the reactive design identity. **RD1** is analagous to **H1** from the theory of designs [6,9], and introduces divergent behaviour: if the predecessor is divergent ($\neg ok$), then a reactive design behaves like $\mathbf{true}_r$ meaning that the only observation is that the trace is extended. **RD2** is identical to **H2** from the theory of designs [6,9]. **RD3** states that $\mathbb{I}_R$ is a right unit of sequential composition. $\mathbf{R}_s$ composes the reactive healthiness conditions and $\mathbf{R3}_h$. We then finally have the healthiness conditions for reactive designs: **SRD** for "stateful reactive designs", and **NSRD** for "normal stateful reactive designs".

Next we define the reactive contract operator.

Definition A.4 (Reactive Contracts).

$$\begin{array}{rcl}
P \vdash Q & \triangleq & (ok \wedge P) \Rightarrow (ok' \wedge Q) \\
P \diamond Q & \triangleq & P \lhd wait' \rhd Q \\
[P \vdash Q \mid R] & \triangleq & \mathbf{R}_s(P \vdash Q \diamond R)
\end{array}$$

A reactive contract is a "reactive design" [7,9]. We construct a UTP design [6] using the design turnstile operator, $P \vdash Q$, and then apply $\mathbf{R}_s$ to the resulting construction. The postcondition of the underlying design is split into two cases for $wait'$ and $\neg wait'$, which indicate whether the observation is quiescent, and correspond to the peri- or postcondition.

Finally, we define the healthiness conditions that specialise our theory to stateful-failure reactive designs.

Definition A.5 (Stateful-Failure Healthiness Conditions).

$$\begin{array}{rcl}
\mathbf{Skip} & \triangleq & [\, \mathbf{true}_r \vdash \mathbf{false} \mid tt = \langle\rangle \wedge st' = st \,] \\
\mathbf{CSP3}(P) & \triangleq & \mathbf{Skip} \fatsemi P \\
\mathbf{CSP4}(P) & \triangleq & P \fatsemi \mathbf{Skip} \\
\mathbf{NCSP}(P) & \triangleq & \mathbf{NSRD} \circ \mathbf{CSP3} \circ \mathbf{CSP4}
\end{array}$$

Skip is similar to $\mathbb{I}_R$, but does not refer to ref in the postcondition. If P is **CSP3** healthy then it cannot refer to ref. If P is **CSP4** healthy then the postcondition cannot refer to ref', but the pericondition can: refusals are only observable when P is quiescent [9,18].